ML LAB: ASSIGNMENT7

# Anandh k 3122 21 5001 009 CSE-A

**A7: Predicting Diabetes using decision tree**

**Aim**: Develop a python program to predict diabetics using Decision Tree Model. Visualize the features from the dataset and interpret the results obtained by the model using Matplotlib library.

The Diabetes prediction dataset is a collection of medical and demographic data from patients, along with their diabetes status (positive or negative). The data includes features such as age, gender, body mass index (BMI), hypertension, heart disease, smoking history, HbA1c level, and blood glucose level. This dataset can be used to build machine learning models to predict diabetes in patients based on their medical history and demographic information. This can be useful for healthcare professionals in identifying patients who may be at risk of developing diabetes and in developing personalized treatment plans. Additionally, the dataset can be used by researchers to explore the relationships between various medical and demographic factors and the likelihood of developing diabetes.

[1]:

**import pandas as pd** **import numpy as np import seaborn as sns**

**from scipy.stats import** zscore **import matplotlib.pyplot as plt from sklearn import** tree

**from sklearn.preprocessing import** MinMaxScaler

**from sklearn.metrics import** classification\_report, roc\_auc\_score,␣

↪confusion\_matrix, ConfusionMatrixDisplay, RocCurveDisplay, roc\_auc\_score

**from sklearn.model\_selection import** train\_test\_split **from sklearn.neighbors import** KNeighborsClassifier **from sklearn.naive\_bayes import** MultinomialNB **from sklearn.tree import** DecisionTreeClassifier **from sklearn import** preprocessing

**from sklearn.decomposition import** PCA

**import scipy.stats**

**from sklearn.metrics import** PrecisionRecallDisplay

**from sklearn.metrics import** confusion\_matrix, ConfusionMatrixDisplay

**from sklearn.neighbors import** KNeighborsClassifier

pd.set\_option('display.float\_format', **lambda** x: '**%.2f**' % x)

%matplotlib inline

C:\Users\Nithi\AppData\Local\Temp\ipykernel\_13632\1512000984.py:1: DeprecationWarning: Pyarrow will become a required dependency of pandas in the next major release of pandas (pandas 3.0),

(to allow more performant data types, such as the Arrow string type, and better interoperability with other libraries)

but was not found to be installed on your system. If this would cause problems for you,

import pandas as pd

# 3 Loading dataset

[2]:

data = pd.read\_csv('C:/Users/ashwi/Downloads/ML Lab/A7/

↪diabetes\_prediction\_dataset.csv')

[3]:

data.head()

1. : gender age hypertension heart\_disease smoking\_history bmi \ 0 Female 80.00 0 1 never 25.19

1 Female 54.00 0 0 No Info 27.32

2 Male 28.00 0 0 never 27.32

3 Female 36.00 0 0 current 23.45

4 Male 76.00 1 1 current 20.14

HbA1c\_level blood\_glucose\_level diabetes 0 6.60 140 0

1 6.60 80 0

2 5.70 158 0

3 5.00 155 0

4 4.80 155 0

[4]:

data.describe()

1. : age hypertension heart\_disease bmi HbA1c\_level \ count 100000.00 100000.00 100000.00 100000.00 100000.00

[5]:

*#Find number of rows and columns*

num\_rows, num\_columns = data.shape

print("Number of rows:", num\_rows) print("Number of columns:", num\_columns)

mean 41.89 0.07 0.04 27.32 5.53

std 22.52 0.26 0.19 6.64 1.07

min 0.08 0.00 0.00 10.01 3.50

25% 24.00 0.00 0.00 23.63 4.80

50% 43.00 0.00 0.00 27.32 5.80

75% 60.00 0.00 0.00 29.58 6.20

max 80.00 1.00 1.00 95.69 9.00

blood\_glucose\_level diabetes count 100000.00 100000.00

mean 138.06 0.09

std 40.71 0.28

min 80.00 0.00

25% 100.00 0.00

50% 140.00 0.00

75% 159.00 0.00

max 300.00 1.00

Number of rows: 100000 Number of columns: 9

[6]:

# 4 Pre-Processing the data

[6]: diabetes

*#Distribution*

data['diabetes'].value\_counts()

0 91500

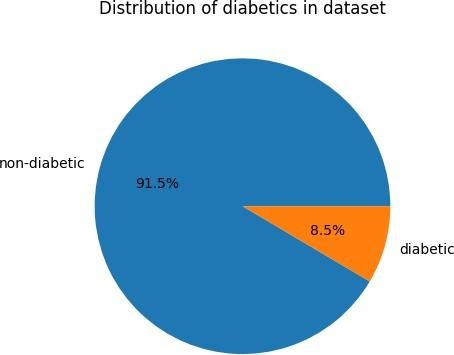
1 8500

Name: count, dtype: int64

[7]:

plt.pie(data['diabetes'].value\_counts(), labels = ['non-diabetic', 'diabetic'], autopct = '**%1.1f%%**')

plt.title("Distribution of diabetics in dataset") plt.show()



[8]:

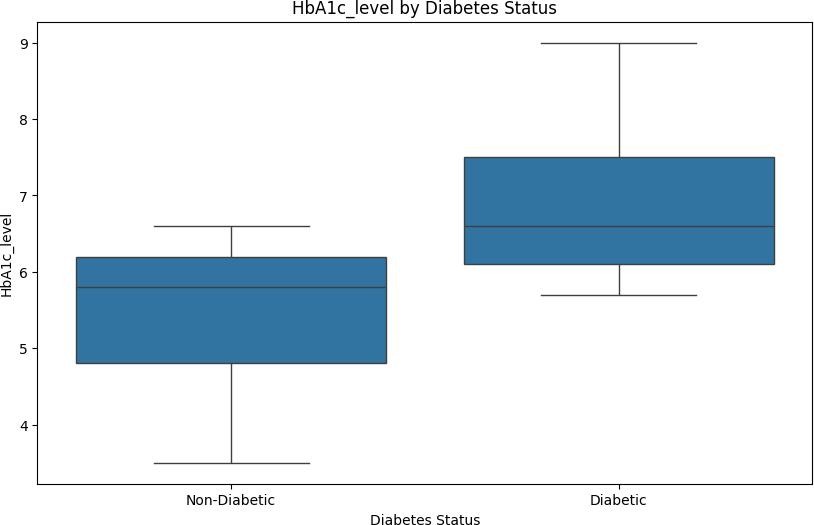
plt.figure(figsize=(10, 6)) sns.boxplot(x='diabetes', y='HbA1c\_level', data=data) plt.title('HbA1c\_level by Diabetes Status') plt.xlabel('Diabetes Status') plt.ylabel('HbA1c\_level')

plt.xticks([0, 1], ['Non-Diabetic', 'Diabetic'])

↪*'diabetes' column encoding*

plt.show()

*# Adjust based on your*␣



[9]:

plt.figure(figsize=(10, 6))

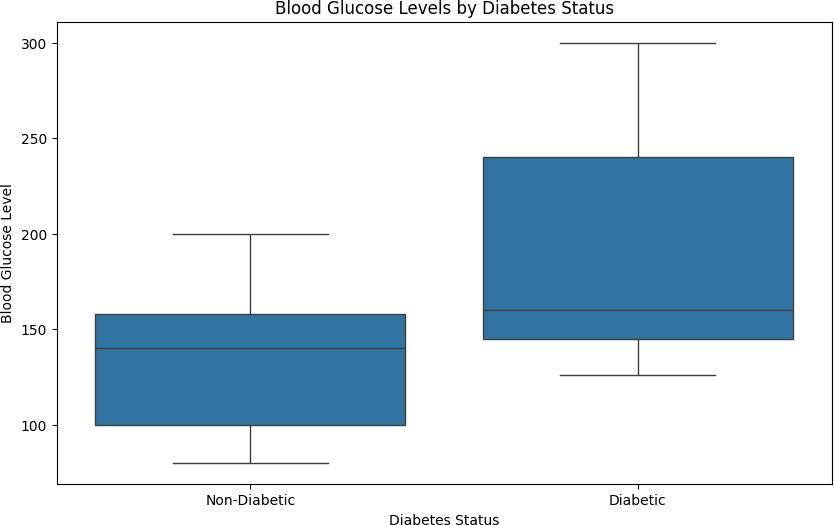
sns.boxplot(x='diabetes', y='blood\_glucose\_level', data=data) plt.title('Blood Glucose Levels by Diabetes Status') plt.xlabel('Diabetes Status')

plt.ylabel('Blood Glucose Level')

plt.xticks([0, 1], ['Non-Diabetic', 'Diabetic']) *# Adjust based on your*␣

↪*'diabetes' column encoding*

plt.show()



[10]:

*#Find Missing Values*

missing\_values = data.isnull().sum().sum() percentage\_missing = (missing\_values / data.shape[0]) \* 100 print("Percentage of missing values:", percentage\_missing)

Percentage of missing values: 0.0

[11]:

*#Convert Categorical variables into binary and numeric*

label\_encoder = preprocessing.LabelEncoder()

data['smoking\_history'] = label\_encoder.fit\_transform(data['smoking\_history']) data['gender'] = label\_encoder.fit\_transform(data['gender'])

data.head()

[11]: gender age hypertension heart\_disease smoking\_history bmi \ 0 0 80.00 0 1 4 25.19

1 0 54.00 0 0 0 27.32

2 1 28.00 0 0 4 27.32

3 0 36.00 0 0 1 23.45

4 1 76.00 1 1 1 20.14

HbA1c\_level blood\_glucose\_level diabetes 0 6.60 140 0

1 6.60 80 0

[12]:

numeric\_data = data.select\_dtypes(include='number') print(numeric\_data)

2 5.70 158 0

3 5.00 155 0

4 4.80 155 0

[13]:

[14]:

*# Calculate Z-scores for numeric columns*

z\_scores = data[numerical\_columns].apply(zscore)

print(z\_scores)

gender age hypertension heart\_disease smoking\_history bmi \ 0 0 80.00 0 1 4 25.19

1 0 54.00 0 0 0 27.32

2 1 28.00 0 0 4 27.32

3 0 36.00 0 0 1 23.45

4 1 76.00 1 1 1 20.14

… … … … … … …

99995 0 80.00 0 0 0 27.32

99996 0 2.00 0 0 0 17.37

99997 1 66.00 0 0 3 27.83

99998 0 24.00 0 0 4 35.42

99999 0 57.00 0 0 1 22.43

HbA1c\_level blood\_glucose\_level diabetes 0 6.60 140 0

1 6.60 80 0

2 5.70 158 0

3 5.00 155 0

4 4.80 155 0

… … … …

99995 6.20 90 0

99996 6.50 100 0

99997 5.70 155 0

99998 4.00 100 0

99999 6.60 90 0

[100000 rows x 9 columns]

*# Selecting the numerical columns (excluding binary columns for hypertension,*␣

↪*heart\_disease, diabetes)*

numerical\_columns = ['age', 'bmi', 'HbA1c\_level','smoking\_history',␣

↪'blood\_glucose\_level']

age bmi HbA1c\_level smoking\_history blood\_glucose\_level 0 1.69 -0.32 1.00 0.96 0.05

[15]:

1 0.54 -0.00 1.00 -1.15 -1.43

2 -0.62 -0.00 0.16 0.96 0.49

3 -0.26 -0.58 -0.49 -0.62 0.42

4 1.52 -1.08 -0.68 -0.62 0.42

… … … … … …

99995 1.69 -0.00 0.63 -1.15 -1.18

99996 -1.77 -1.50 0.91 -1.15 -0.93

99997 1.07 0.08 0.16 0.43 0.42

99998 -0.79 1.22 -1.43 0.96 -0.93

99999 0.67 -0.74 1.00 -0.62 -1.18

[100000 rows x 5 columns]

*# Define threshold for outlier detection (e.g., Z-Score > 3)*

threshold = 3

*# Find outliers*

outliers = data[z\_scores > threshold] print(outliers)

gender age hypertension heart\_disease smoking\_history bmi \

1. NaN NaN NaN NaN NaN NaN
2. NaN NaN NaN NaN NaN NaN
3. NaN NaN NaN NaN NaN NaN
4. NaN NaN NaN NaN NaN NaN
5. NaN NaN NaN NaN NaN NaN

… … … … … … …

99995 NaN NaN NaN NaN NaN NaN

99996 NaN NaN NaN NaN NaN NaN

99997 NaN NaN NaN NaN NaN NaN

99998 NaN NaN NaN NaN NaN NaN

99999 NaN NaN NaN NaN NaN NaN

HbA1c\_level blood\_glucose\_level diabetes

1. NaN NaN NaN
2. NaN NaN NaN
3. NaN NaN NaN
4. NaN NaN NaN
5. NaN NaN NaN

… 99995

…

NaN

… …

NaN

NaN

99996 NaN NaN NaN

99997 NaN NaN NaN

99998 NaN NaN NaN

99999 NaN NaN NaN

[100000 rows x 9 columns]

[16]:

*# Count outliers*

outliers\_count = (z\_scores.abs() > threshold).sum().sum()

print("Number of outliers:", outliers\_count)

Number of outliers: 4012

[17]:

*# normalization*

*# Initialize the MinMaxScaler*

scaler = MinMaxScaler()

*# Fit the scaler to the data and transform it*

data[numerical\_columns] = scaler.fit\_transform(data[numerical\_columns])

*# Display the first few rows to verify the normalization*

print(data.head())

[18]:

gender age hypertension heart\_disease smoking\_history bmi \ 0 0 1.00 0 1 0.80 0.18

1 0 0.67 0 0 0.00 0.20

2 1 0.35 0 0 0.80 0.20

3 0 0.45 0 0 0.20 0.16

4 1 0.95 1 1 0.20 0.12

HbA1c\_level blood\_glucose\_level diabetes 0 0.56 0.27 0

1 0.56 0.00 0

2 0.40 0.35 0

3 0.27 0.34 0

4 0.24 0.34 0

data.head()

[18]: gender age hypertension heart\_disease smoking\_history bmi \ 0 0 1.00 0 1 0.80 0.18

1 0 0.67 0 0 0.00 0.20

2 1 0.35 0 0 0.80 0.20

3 0 0.45 0 0 0.20 0.16

4 1 0.95 1 1 0.20 0.12

HbA1c\_level blood\_glucose\_level diabetes 0 0.56 0.27 0

1 0.56 0.00 0

2 0.40 0.35 0

3 0.27 0.34 0

4 0.24 0.34 0

[19]:

*# Calculate the correlation matrix*

corr\_matrix = data.corr()

*# Plot the heatmap*

plt.figure(figsize=(10, 8))

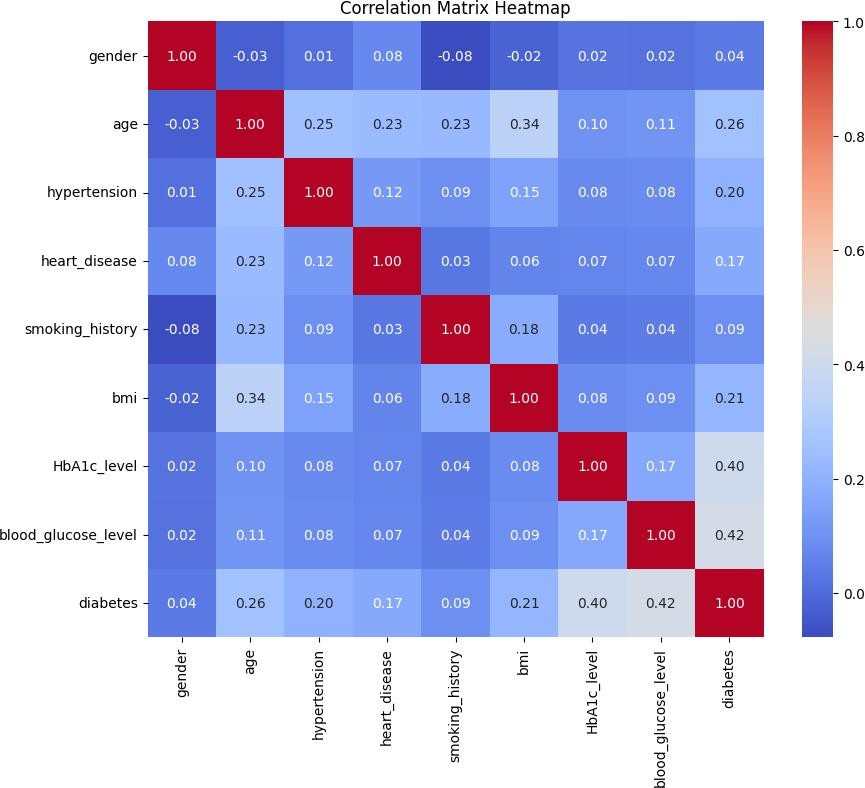
*# Set the figure size as desired*

sns.heatmap(corr\_matrix, annot=**True**, fmt=".2f", cmap='coolwarm', cbar=**True**,␣

↪square=**True**)

plt.title('Correlation Matrix Heatmap') plt.show()

# Exploratory Data Analysis



[20]:

# Split the data into training, testing and validation sets

[21]:

X = data.drop(columns=['diabetes'], axis=1) y = data['diabetes']

X\_train\_raw, X\_test\_raw, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.

↪30, random\_state=42)

[22]:

X\_train = pd.DataFrame((X\_train\_raw), columns=X\_train\_raw.columns) X\_test = pd.DataFrame((X\_test\_raw), columns=X\_test\_raw.columns) X\_test.head()

1. : gender age hypertension heart\_disease smoking\_history bmi \ 75721 0 0.16 0 0 0.00 0.13

80184 0 0.04 0 0 0.00 0.13

19864 1 0.79 0 0 0.60 0.18

76699 0 0.02 0 0 0.80 0.09

92991 0 0.41 0 0 1.00 0.35

HbA1c\_level blood\_glucose\_level 75721 0.42 0.21

80184 0.27 0.30

19864 0.00 0.55

76699 0.47 0.21

92991 0.49 0.55

[23]:

# Train the model

1. : PCA(random\_state=42)

pca = PCA(random\_state=42) pca.fit(X\_train)

[24]:

*# Calculate the cumulative explained variance ratio*

cumulative\_variance = pca.explained\_variance\_ratio\_.cumsum()

*# Find the number of components needed for 90% variance*

n\_components\_90 = (cumulative\_variance <= 0.90).sum()

print("Number of components to explain 90% variance:", n\_components\_90)

Number of components to explain 90% variance: 5

[25]:

pca = PCA(n\_components=5, random\_state=42) pca.fit(X\_train)

X\_train\_PCA = pd.DataFrame(pca.transform(X\_train)) X\_test\_PCA = pd.DataFrame(pca.transform(X\_test))

[26]:

X\_train\_PCA.columns = [str(column\_name) **for** column\_name **in** X\_train\_PCA.columns] X\_test\_PCA.columns = [str(column\_name) **for** column\_name **in** X\_test\_PCA.columns]

[27]:

X\_train\_PCA.head()

[27]: 0 1 2 3 4

0 0.58 0.06 -0.01 -0.11 -0.06

1 0.51 0.47 -0.09 -0.19 -0.33

2 -0.37 -0.39 -0.13 0.07 -0.26

3 -0.38 -0.22 0.20 -0.17 0.17

4 0.66 -0.46 -0.13 0.12 0.02

# Test the model

[28]:

model = tree.DecisionTreeClassifier()

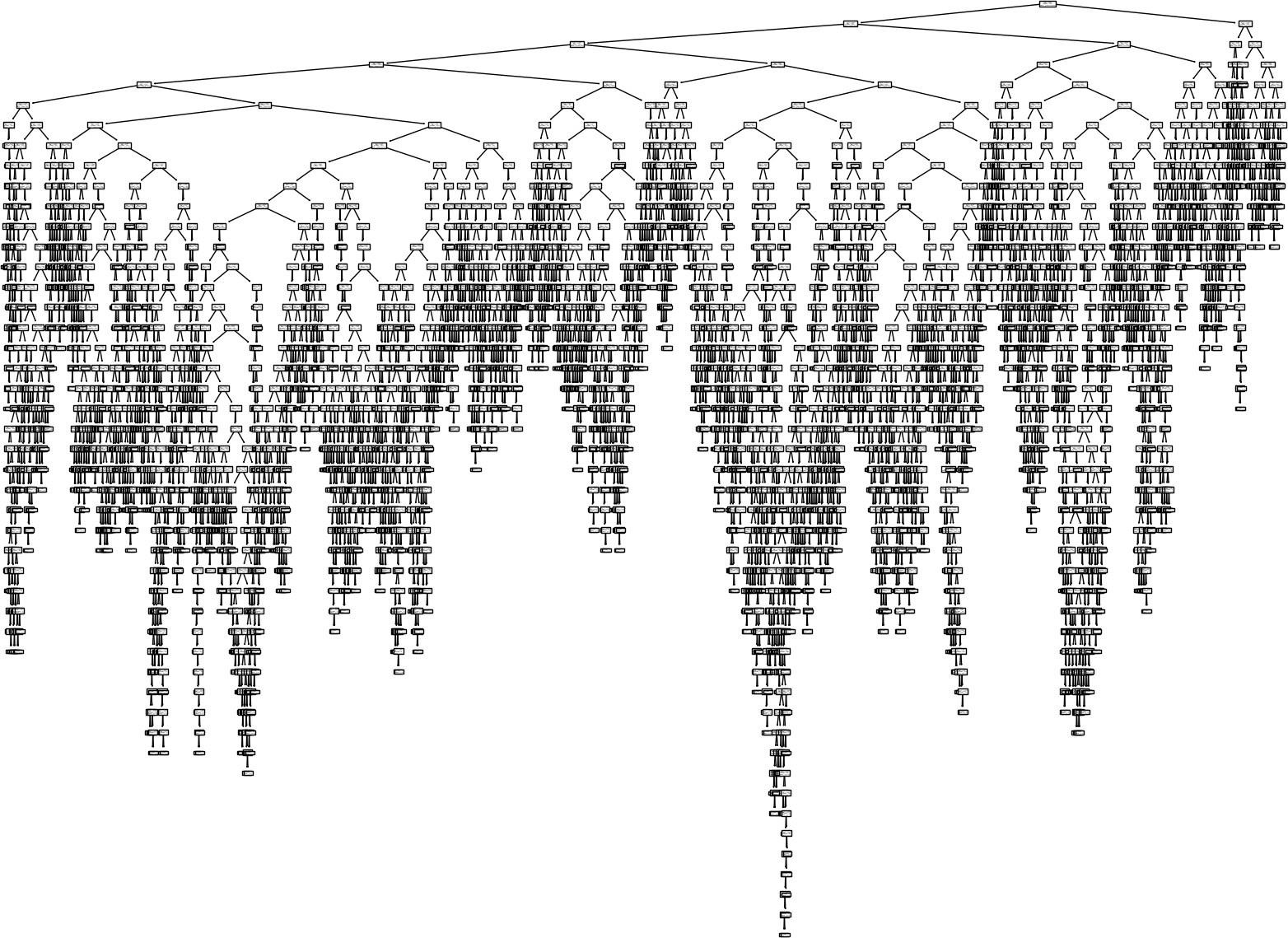
[29]:

model.fit(X\_train\_PCA, y\_train)

[29]: DecisionTreeClassifier()

[30]:

fig = plt.figure(figsize=(20,15)) tree.plot\_tree(model);

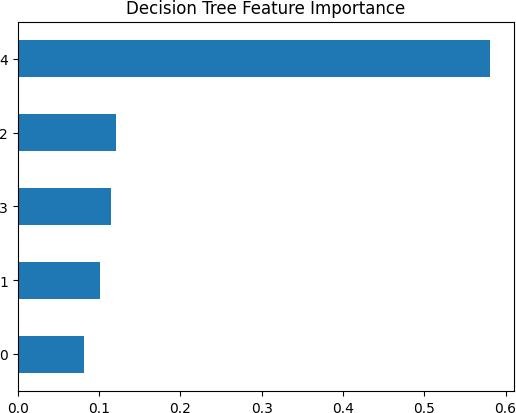


[31]:

feature\_importances = pd.Series(model.feature\_importances\_, model.

↪feature\_names\_in\_).sort\_values()

feature\_importances.plot.barh(); plt.title('Decision Tree Feature Importance');



[32]:

y\_pred\_test = model.predict(X\_test\_PCA) y\_pred\_train= model.predict(X\_train\_PCA)

[33]:

train\_decision\_tree\_report = classification\_report(y\_train, y\_pred\_train) print(train\_decision\_tree\_report)

# Measure the performance of the trained model

|  |  |  |  |
| --- | --- | --- | --- |
| precision | recall | f1-score | support |
| 0 1.00 | 1.00 | 1.00 | 64047 |
| 1 1.00 | 0.99 | 1.00 | 5953 |

accuracy 1.00 70000

macro avg 1.00 1.00 1.00 70000

weighted avg 1.00 1.00 1.00 70000

[34]:

test\_decision\_tree\_report = classification\_report(y\_test, y\_pred\_test) print(test\_decision\_tree\_report)

|  |  |  |  |
| --- | --- | --- | --- |
| precision | recall | f1-score | support |
| 0 0.96 | 0.97 | 0.97 | 27453 |
| 1 0.62 | 0.62 | 0.62 | 2547 |

accuracy 0.94 30000

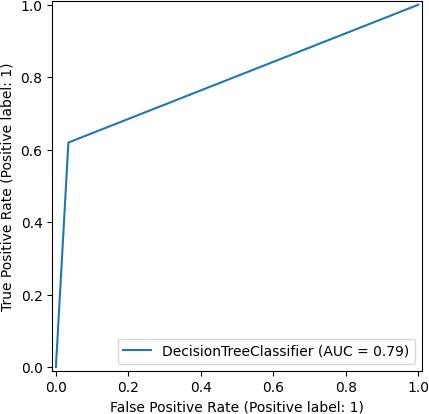
macro avg 0.79 0.79 0.79 30000

weighted avg 0.94 0.94 0.94 30000

[35]:

RocCurveDisplay.from\_estimator(model, X\_test\_PCA, y\_test) plt.show()

# Represent the results using graphs



[36]:

**from sklearn.tree import** DecisionTreeClassifier

**from sklearn.metrics import** accuracy\_score

**from sklearn.decomposition import** PCA

**from sklearn.model\_selection import** train\_test\_split

**import numpy as np**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_train\_PCA, y\_train,␣

↪test\_size=0.2, random\_state=42)

dt\_entropy = DecisionTreeClassifier(criterion='entropy', random\_state=42) dt\_entropy.fit(X\_train, y\_train)

y\_pred\_entropy = dt\_entropy.predict(X\_test) accuracy\_entropy = accuracy\_score(y\_test, y\_pred\_entropy)

print("Accuracy using Entropy impurity measure with PCA:", accuracy\_entropy)

dt\_gini = DecisionTreeClassifier(criterion='gini', random\_state=42) dt\_gini.fit(X\_train, y\_train)

y\_pred\_gini = dt\_gini.predict(X\_test)

accuracy\_gini = accuracy\_score(y\_test, y\_pred\_gini)

print("Accuracy using Gini-index impurity measure with PCA:", accuracy\_gini)

[37]:

[44]:

**from sklearn.tree import** DecisionTreeClassifier

**from sklearn.metrics import** accuracy\_score

**from sklearn.model\_selection import** train\_test\_split

**import pandas as pd**

dt\_entropy = DecisionTreeClassifier(criterion='entropy', random\_state=42) dt\_entropy.fit(X\_train\_raw, y\_train)

Accuracy using Entropy impurity measure with PCA: 0.9377857142857143 Accuracy using Gini-index impurity measure with PCA: 0.9399285714285714

**from sklearn.tree import** DecisionTreeClassifier

**from sklearn.metrics import** accuracy\_score

dt\_classifier = DecisionTreeClassifier(random\_state=42) dt\_classifier.fit(X\_train, y\_train)

y\_pred = dt\_classifier.predict(X\_test)

accuracy\_without\_pca = accuracy\_score(y\_test, y\_pred) print("Accuracy without PCA:", accuracy\_without\_pca)

Accuracy without PCA: 0.9399285714285714

y\_pred\_entropy = dt\_entropy.predict(X\_test\_raw) accuracy\_entropy = accuracy\_score(y\_test, y\_pred\_entropy)

print("Accuracy using Entropy impurity measure without PCA:", accuracy\_entropy)

dt\_gini = DecisionTreeClassifier(criterion='gini', random\_state=42) dt\_gini.fit(X\_train\_raw, y\_train)

y\_pred\_gini = dt\_gini.predict(X\_test\_raw) accuracy\_gini = accuracy\_score(y\_test, y\_pred\_gini)

print("Accuracy using Gini-index impurity measure without PCA:", accuracy\_gini)

Accuracy using Entropy impurity measure without PCA: 0.9542666666666667 Accuracy using Gini-index impurity measure without PCA: 0.9530333333333333

[43]: X = data.drop(columns=['diabetes'], axis=1) y = data['diabetes']

X\_train\_raw, X\_test\_raw, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.

↪3, random\_state=42)

print("X\_train\_raw shape:", X\_train\_raw.shape) print("y\_train shape:", y\_train.shape) print("X\_test\_raw shape:", X\_test\_raw.shape) print("y\_test shape:", y\_test.shape)

dt\_entropy = DecisionTreeClassifier(criterion='entropy', random\_state=42) dt\_entropy.fit(X\_train\_raw, y\_train)

y\_pred\_entropy = dt\_entropy.predict(X\_test\_raw) accuracy\_entropy = accuracy\_score(y\_test, y\_pred\_entropy)

print("Accuracy using Entropy impurity measure without PCA:", accuracy\_entropy)

dt\_gini = DecisionTreeClassifier(criterion='gini', random\_state=42) dt\_gini.fit(X\_train\_raw, y\_train)

y\_pred\_gini = dt\_gini.predict(X\_test\_raw) accuracy\_gini = accuracy\_score(y\_test, y\_pred\_gini)

print("Accuracy using Gini-index impurity measure without PCA:", accuracy\_gini)

X\_train\_raw shape: (70000, 8)

y\_train shape: (70000,)

X\_test\_raw shape: (30000, 8)

y\_test shape: (30000,)

Accuracy using Entropy impurity measure without PCA: 0.9542666666666667 Accuracy using Gini-index impurity measure without PCA: 0.9530333333333333

[45]: **from sklearn.metrics import** roc\_curve, auc

## import matplotlib.pyplot as plt

y\_pred\_entropy\_no\_pca = dt\_entropy.predict(X\_test\_raw) y\_pred\_gini\_no\_pca = dt\_gini.predict(X\_test\_raw)

fpr\_entropy\_pca, tpr\_entropy\_pca, \_ = roc\_curve(y\_test, y\_pred\_entropy) roc\_auc\_entropy\_pca = auc(fpr\_entropy\_pca, tpr\_entropy\_pca)

fpr\_gini\_pca, tpr\_gini\_pca, \_ = roc\_curve(y\_test, y\_pred\_gini) roc\_auc\_gini\_pca = auc(fpr\_gini\_pca, tpr\_gini\_pca)

fpr\_entropy\_no\_pca, tpr\_entropy\_no\_pca, \_ = roc\_curve(y\_test,␣

↪y\_pred\_entropy\_no\_pca)

roc\_auc\_entropy\_no\_pca = auc(fpr\_entropy\_no\_pca, tpr\_entropy\_no\_pca)

fpr\_gini\_no\_pca, tpr\_gini\_no\_pca, \_ = roc\_curve(y\_test, y\_pred\_gini\_no\_pca) roc\_auc\_gini\_no\_pca = auc(fpr\_gini\_no\_pca, tpr\_gini\_no\_pca)

*# Plot ROC curve*

plt.figure(figsize=(8, 6))

plt.plot(fpr\_entropy\_pca, tpr\_entropy\_pca, color='darkorange', lw=2, label='ROC␣

↪curve (Entropy with PCA) (area = **%0.2f**)' % roc\_auc\_entropy\_pca) plt.plot(fpr\_gini\_pca, tpr\_gini\_pca, color='green', lw=2, label='ROC curve␣

↪(Gini with PCA) (area = **%0.2f**)' % roc\_auc\_gini\_pca) plt.plot(fpr\_entropy\_no\_pca, tpr\_entropy\_no\_pca, color='blue', lw=2, label='ROC␣

↪curve (Entropy without PCA) (area = **%0.2f**)' % roc\_auc\_entropy\_no\_pca)

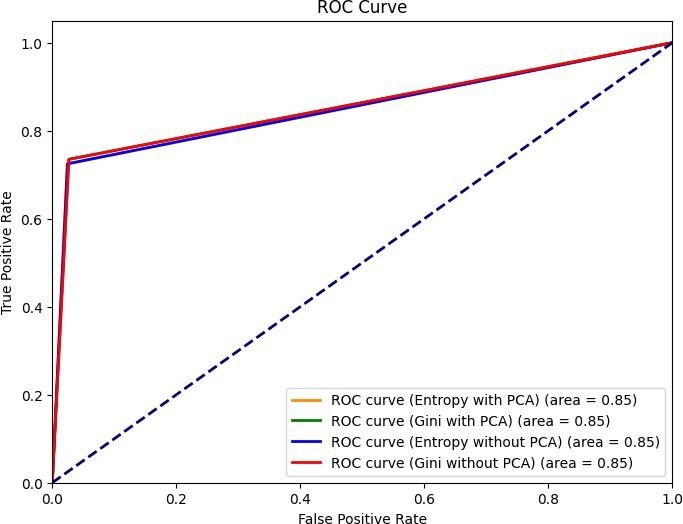
plt.plot(fpr\_gini\_no\_pca, tpr\_gini\_no\_pca, color='red', lw=2, label='ROC curve␣

↪(Gini without PCA) (area = **%0.2f**)' % roc\_auc\_gini\_no\_pca) plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05]) plt.xlabel('False Positive Rate') plt.ylabel('True Positive Rate') plt.title('ROC Curve') plt.legend(loc="lower right")

plt.show()



[46]:

print("X\_train\_raw shape:", X\_train\_raw.shape) print("y\_train shape:", y\_train.shape) print("X\_test\_raw shape:", X\_test\_raw.shape) print("y\_test shape:", y\_test.shape)

X\_train\_raw shape: (70000, 8)

y\_train shape: (70000,)

X\_test\_raw shape: (30000, 8)

y\_test shape: (30000,)

# Inference

1.Accuracy using Entropy impurity measure without PCA: 0.9542666666666667 Accuracy using Gini-index impurity measure without PCA: 0.9530333333333333

2.Accuracy using Entropy impurity measure with PCA: 0.9377857142857143

Accuracy using Gini-index impurity measure with PCA: 0.9399285714285714

As we can clearly see that the accuracy of the decision tree classifier using Gini-index and Entropy using PCA gives less accuracy than without using PCA.

# Learning Outcomes

* 1. Applying standardization and other pre-processing techniques.
  2. Splitting the data according to PCA.
  3. Understanding Decision Tree classifier Algorithm.
  4. Implementing decision tree classifier model.
  5. Implementing the decision tree model with using PCA and without using PCA. GITHUB LINK:

[https://github.com/Anandh-007/Machine-learning-lab/tree/main/ML\_A7](file:///C:\Users\anand\Downloads\as.docx)